CSE 232 Spring 2018

Programming Project #1

Assignment Overview

This project focuses on some mathematical manipulation. It is worth 5 points (0.5% of your overall grade). It is due Monday, January 22nd before midnight.

The Problem

Hey, you're studying computer science. That likely means a job with a decent salary is in your future. If so, what's the first thing you want to buy when you graduate? A car of course! Let's write a program to help you calculate your monthly payment so you know what you are up against.

Some Background

The hard part of doing this is getting two things correct:

- the correct order of calculation
- the formula for compound interest.

Here is the formula for compound interest (you can look here for more info https://www.wikihow.com/Calculate-Auto-Loan-Payments)

monthly_payment =
Final_price_minus_downpayment * ((monthly_rate * (1 + monthly_rate)$^{num\_months}$ / (1 + monthly_rate)$^{num\_months}$ - 1)

Program Specifications

Your program will do the following:

1. Take as input six double values (in this order, as indicated):
    a. car price
    b. sales tax rate (as a decimal number, i.e. 6% is 0.06)
    c. downpayment (a dollar amount)
    d. title_and_fees (fees plus title, a dollar amount)
    e. yearly interest rate (as a decimal number, i.e. 7% would be 0.07)
    f. number of months the loan will last (the loan's duration)
2. Print a single result, your monthly payment
3. The order matters here:
    a. you multiply the car price times the sales tax (sales tax is on the original price)
    b. you then add to the price the sales_tax amount plus the titles and fees. This is your total cost.
    c. now remove any downpayment from the total cost
    d. take the yearly interest rate and turn it into the monthly_rate by dividing by 12.0
    e. calculate the monthly payment using the compound interest formula

Deliverables

proj01/proj01.cpp . The name of the directory is proj01, the name file you turn in should be exactly proj01.cpp in a directory names proj01. It will be checked upon handing it in to

Mimir for the first test case. Just like the lab, you must click on proj01 within "Project 01 – buying a car" to submit the file. Not the file proj01.cpp, the directory proj01

Assignment Notes:

1. We might as well try to make the output somewhat readable. You can look this up in the text or on the internet, but you can use the following modifiers that affect how numbers print.
   a. `cout << fixed`
      Elements will be printed as floating point numbers (ex 123.456). **Use this for the project.**
   b. `cout << scientific`
      Elements will be printed in scientific notation (ex 1.23456×10^2). This is an alternative but ***not what we want*** for this project.
   c. `cout << setprecision(2)` Floating point numbers will have 2 values after the decimal point and will be rounded, (123.46). To make this work we need another include, and that is
      `#include<iomanip>`
      Provide the `include` and use `setprecision(2)` for this project.
   d. Thus `cout << fixed << setprecision(2) << 123.4567 << endl;` will print 123.46
2. The following statement will read two variables off of the same, space separated line. It is an example of chaining input:

```
…
double d1, d2;
cin >> d1 >> d2;
…
```

   You can also do it on separate lines. Your choice

```
…
cin >> d1;
cin >> d2;
…
```

3. If you `#include<cmath>` you get access to the `std::pow` function. The `std::pow` function takes as arguments two number: the first a number to raise to some power and the second the power to which the first is raised. It returns a `double`. Thus `std::pow(5,2)` returns 25.0 . Note that pow can take either floating point or integer arguments, but it returns a `double` (but see http://en.cppreference.com/w/cpp/numeric/math/pow for full details)
4. There are 4 tests provided. The first is a file-exists test to make sure you got the directory issue correct, the remaining three are input-output tests.
5. You ***do not*** have to check for bad input values. In general, we will explicitly indicate the errors we are looking for, but for now we are not checking for input errors.
6. It gets irritating to type in 6 numbers every time you test. To get around this, you can use a handy trick off the command line. You can create a file with the 6 input values (on a single line, they need to be space separated, on 6 different lines is also fine). You can **redirect** the file to your main program. With that file in the same directory as the compiled `a.out`, you can do:

```
./a.out < fileOfInput.txt
```

This will automatically feed the input to the `cin` statement and produce the output. Makes it easier to test things. An example `input.txt` is provided in the project directory.

Getting Started

1. In Mimir, I create the directory `proj01` and place in it an empty file (no contents) with the correct name, `proj01`. You can update the file contents there.
2. If you are in a CSE lab or on x2go (or whatever your environment), then bring up an editor and a terminal, create a project directory of any name (I would suggest the Desktop as that is easiest to work with) and create proj01.cpp
3. Write your code and compile it.
4. Prompt for some of the values and print them back out, just to check yourself.
5. Check that your calculations are right (as indicated above).
6. If you develop on x2go (or wherever), the easiest way at this point to check yourself on Mimir is to copy and pasted rom your editor to the empty file on Mimir and submit.
    a. You don't have to pass all the tests the first time! You can add more information and pass more of the tests as you progress. Do things incrementally.
7. Now you enter a cycle of edit-run to incrementally develop your program.
8. If a version you submit and test on Mimir passes all the tests, you are done. If time runs out and you didn't pass all the tests, then however many you passed indicates what grade you got for the project. Though it doesn't matter much now as this is relatively easy, at the end you do the best you can and pass as many tests as possible. However, ***you always know*** how you are doing and that is the advantage of Mimir.
9. ***Remember***: In the end, it only matters that it compiles and runs on Mimir. If it doesn't compile there, then it doesn't compile at all (no matter what else you did on whatever environment you used). Mimir is the last word.